Guide for Creating Your Own Reporting Measures

Measure Intent

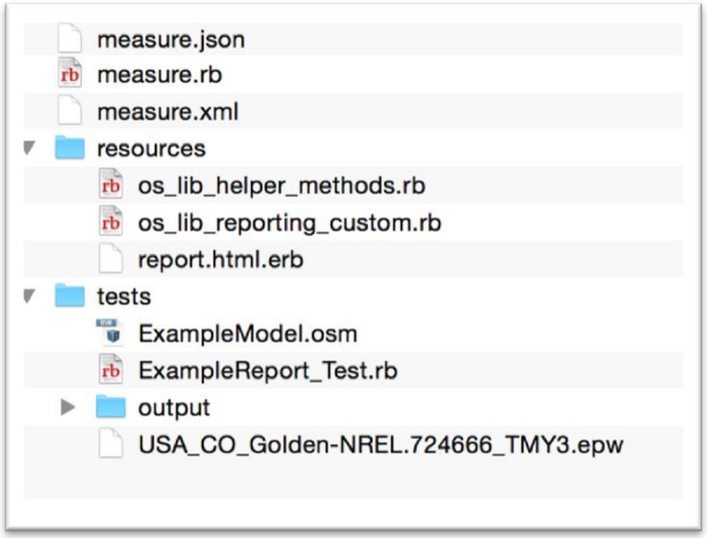
- Provide a clean simple example for reporting tabular data and charts for model inputs and simulation results.
- Minimize coding that has to be done by measure authors.
  - Copy and alter example section and table methods in the measure to meet your own use case.
- Provide robust performance with consistent look with measures by other authors.

How the Measure Works

- The measure uses bootstrap to produce a clean tables as well as the table of contents.
- dimplejs which uses d3 is used to generate basic charts.
- report.html.erb is javascript code called by measure.rb to generate the html report.
- You don’t have to worry about anything above. All you need to do is change os_lib_reporting_custom.rb as described on the following slides.

Making Your Own Reporting Measure

- Download this measure through the the OpenStudio application or PAT.
- Then use the “x2” button while this measure is selected to make a copy of the measure in your “My Measures” directory.
  - You want to use the GUI to copy so your measure gets its own unique ID.
- While copying your measure take time to give it an updated name, description, and modeler description.
- The only file you need to change is os_lib_reporting.rb in the measure’s resources directory.
- See the OpenStudio Measure Writing Guide for directions on running tests.
  - While you can test in the GUI, testing outside of the GUI allows you to quickly test without having to re-run the simulation.
  - The existing ExampleReport_Test.rb will work on your modified measure without any changes. The exception would be if you you need to change out the Example Model if it doesn’t have the objects you need for your report.



```
measure.json
measure.rb
measure.xml
▾ resources
    os_lib_helper_methods.rb
    os_lib_reporting_custom.rb
    report.html.erb
▾ tests
    ExampleModel.osm
    ExampleReport_Test.rb
  ▸ output
    USA_CO_Golden-NREL.724666_TMY3.epw
```

What’s in os_lib_reporting.rb

- Everywhere you see a line start with “def some_name” that is defining a new method.
- The first method is “def setup” you don’t need to change this. This is used by measure.rb to get the model, the idf file, and the SQL results.
- There are two other kinds of methods in the file. They have a name that either end with “section” or “table”.
- If the method name ends with “section” it defines a section for the report.
  - A “section” method may also define a table, or the tables can be in the own “table” method and then referred to by the “section” method.
  - The naming of the “section” method is important. Having “section” in the name is what tells measure.rb that this section exists. It will have a “bool” (checkbox) argument allowing someone using your measure to turn sections on or off.
  - The order of the section methods determines the order of the sections in the report.
- If the method name ends with “table” then it defines a table. This method should be referred to by a “section” measure.

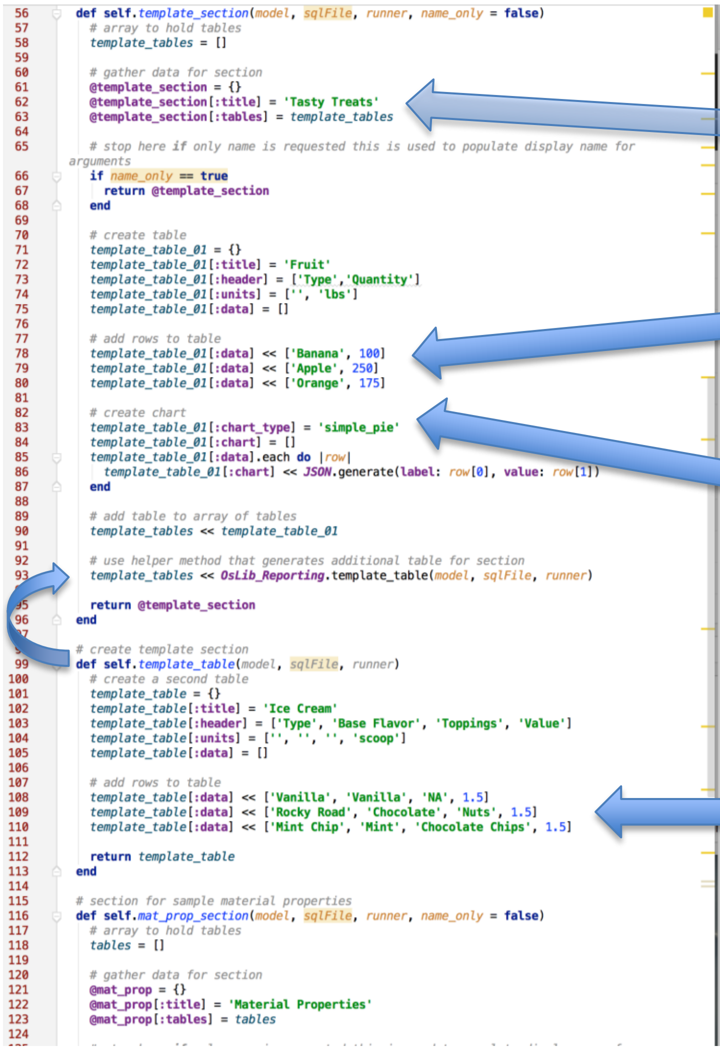
Section Method

- The section measure should always have the same arguments
  - *(model*, *sqlFile*, *runner*, *name_only* = false)
- The first part of the method defines the :title and an empty array of :tables that will be populated later
- Next there is code that stops if name_only = true
  - This is used to get the display name for the user arguments.
- Then one or more tables are added.
  - The code for the table can be in the section or can be in a table method
- "return" at the end of the method passes the data for the section back to measure.rb which in turns passes it to report.html.erb which makes the erb file.

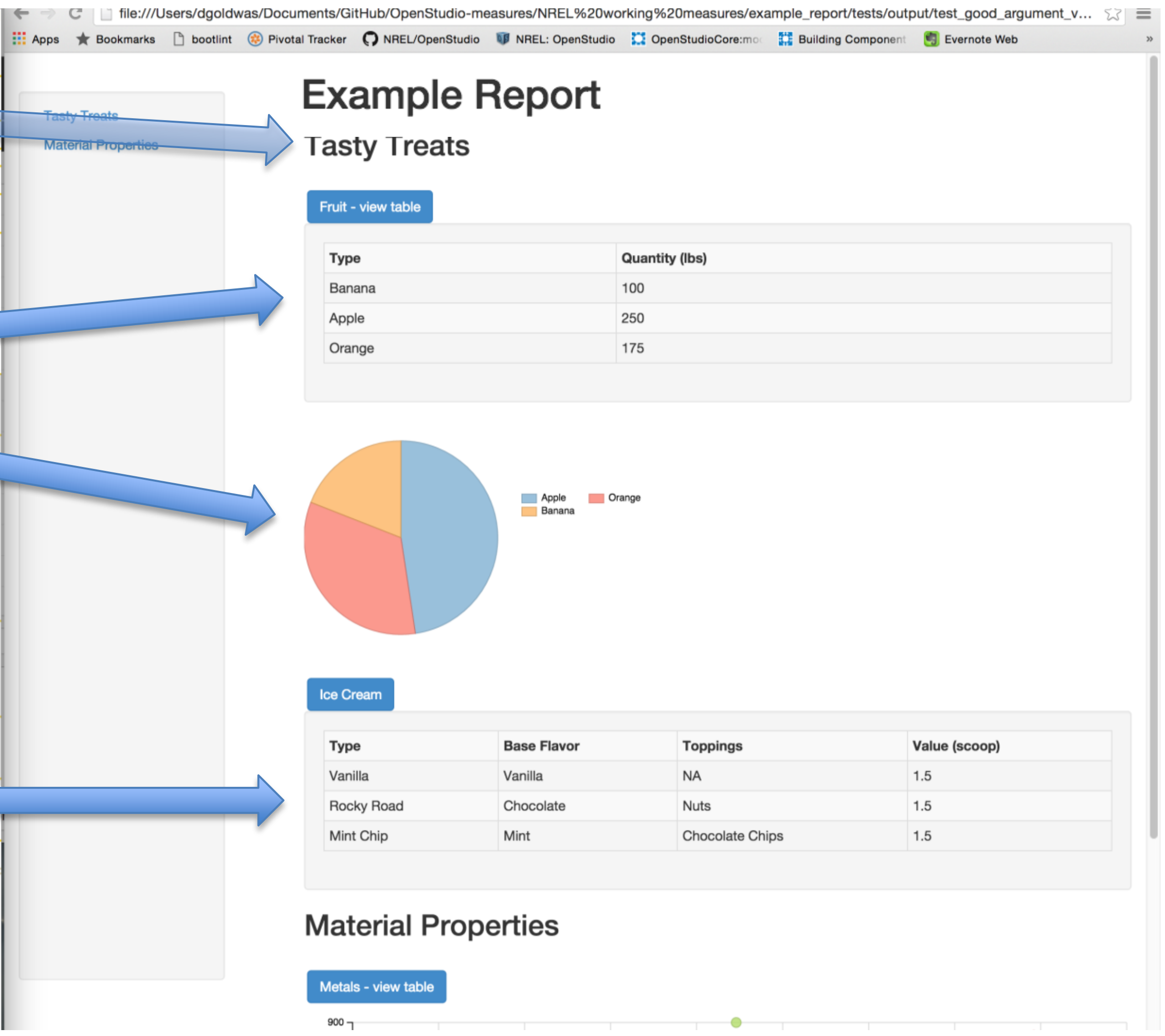
Table Method

- A table section is required to have a :title, :header, and :data.
- :units are optional as is :chart
- You can have as many rows and columns as you want in your table.
  - Make sure the heater, units, and data all have the same number of columns.
- Data for a table and chart can come from the model, the idf, file, the simulation results, an external data source (such as a csv file) or as in the example can be hard coded in the method.
- If you make a chart, keep in mind that different charts take different kinds of data. This measure uses “simple_pie” and “scatter”.
  - See the standard OpenStudio Results measure for examples of more chart types.

Code and Results side by side



```ruby
def self.template_section(model, sqlFile, runner, name_only = false)
  # array to hold tables
  template_tables = []

  # gather data for section
  @template_section = {}
  @template_section[:title] = 'Tasty Treats'
  @template_section[:tables] = template_tables

  # stop here if only name is requested this is used to populate display name for
arguments
  if name_only == true
    return @template_section
  end

  # create table
  template_table_01 = {}
  template_table_01[:title] = 'Fruit'
  template_table_01[:header] = ['Type','Quantity']
  template_table_01[:units] = ['', 'lbs']
  template_table_01[:data] = []

  # add rows to table
  template_table_01[:data] << ['Banana', 100]
  template_table_01[:data] << ['Apple', 250]
  template_table_01[:data] << ['Orange', 175]

  # create chart
  template_table_01[:chart_type] = 'simple_pie'
  template_table_01[:chart] = []
  template_table_01[:data].each do |row|
    template_table_01[:chart] << JSON.generate(label: row[0], value: row[1])
  end

  # add table to array of tables
  template_tables << template_table_01

  # use helper method that generates additional table for section
  template_tables << OsLib_Reporting.template_table(model, sqlFile, runner)

  return @template_section
end

# create template section
def self.template_table(model, sqlFile, runner)
  # create a second table
  template_table = {}
  template_table[:title] = 'Ice Cream'
  template_table[:header] = ['Type', 'Base Flavor', 'Toppings', 'Value']
  template_table[:units] = ['', '', '', 'scoop']
  template_table[:data] = []

  # add rows to table
  template_table[:data] << ['Vanilla', 'Vanilla', 'NA', 1.5]
  template_table[:data] << ['Rocky Road', 'Chocolate', 'Nuts', 1.5]
  template_table[:data] << ['Mint Chip', 'Mint', 'Chocolate Chips', 1.5]

  return template_table
end

# section for sample material properties
def self.mat_prop_section(model, sqlFile, runner, name_only = false)
  # array to hold tables
  tables = []

  # gather data for section
  @mat_prop = {}
  @mat_prop[:title] = 'Material Properties'
  @mat_prop[:tables] = tables
```



Example Report

Tasty Treats

Fruit - view table

Type	Quantity (lbs)
Banana	100
Apple	250
Orange	175

Ice Cream

Type	Base Flavor	Toppings	Value (scoop)
Vanilla	Vanilla	NA	1.5
Rocky Road	Chocolate	Nuts	1.5
Mint Chip	Mint	Chocolate Chips	1.5

Material Properties

Metals - view table

Getting Real Data

- The general_building_information_table method at the end of os_lib_reporting_custom.rb demostrates a number of ways to get real dataGet the name of the building from the model.
  - *model*.getBuilding.name.to_s
- There are some methods to get data right out of the SQL file like total site energy
  - *sqlFile*.netSiteEnergy.get
- Here is an example where a query had to be made into the tabular SQL data
  - *query* = 'SELECT Value FROM tabulardatawithstrings WHERE '
    query << "ReportName='AnnualBuildingUtilityPerformanceSummary' and "
    query << "ReportForString='Entire Facility' and "
    query << "TableName='Building Area' and "
    query << "RowName='Total Building Area' and "
    query << "ColumnName='Area' and "
    query << "Units='m2';"
    query_results = *sqlFile*.execAndReturnFirstDouble(*query*)
- Any data you get out of the model, IDF file, or SQL database will be in SI units. Here is an example of converting it to IP.
  - *OpenStudio*.convert(*sqlFile*.netSiteEnergy.get, *‘GJ’*,’kBtu’).get

Limitations and Known Issues

- The example measure is setup to be a reporting measure that runs after the simulation is done. If you are only interested in model inputs you could adapt a copy of this measure to run as a model measure. You could then use it in apply measures now in the OpenStudio application.
- The .erb file is currently only setup for a few types of dimple charts, and only a few of those are exercised. Look at the OpenStudio Results measure on the BCL for example of other chart types
  - simple_pie
  - vertical_stacked_bar
  - vertical_grouped_bar
  - scatter
  - vertical_grouped_bar_with_comp_line
  - multi_step_line_grid